pipeline 59. Note that it is not necessary for the same offset to be added to each coordinate rendered by a particular pipeline 56-59. For example, one of the pipelines 56-59 could be configured to add the value of .1 to each x-coordinate value rendered by the one pipeline 56-59 and to add the value of .2 to each y-coordinate value and z-coordinate value rendered by the one pipeline 56-59.

The graphical data in frame buffers 66-69 is transmitted to compositor 76, which forms a single graphical representation of the object 284 based on each of the graphical representations from frame buffers 66-69. In this regard, the compositor 76 averages or blends into a single color value the color values of each pixel from frame buffers 66-69 having the same screen relative coordinate values. Each color value calculated by the compositor 76 is then assigned to the pixel having the same coordinate values as the pixels that were averaged or blended to form the color value calculated by the compositor 76.

As an example, assume that color values stored in frame buffers 66-69 for the pixel having the coordinate values (1000, 1000, 0) are a, b, c, and d, respectively, in which a, b, c, and d represent four different numerical values. In this example, the compositor 76 may calculate a new color value, n, based on the following equation: $n = (a + b + c + d)/4$. This new color value, n, is then transmitted to display device 83 as the color value for the pixel having coordinates (1000, 1000, 0). Note that a different algorithm may be used to calculate the new color value and that different weightings may be applied to the values being averaged.

By performing the above-described process for each pixel represented in frame buffers 66-69, the compositor 76 produces graphical data defining a jitter enhanced image of the 3D object 284. This data is rendered to the display device 83 to display the jitter enhanced image of the object 284.

It should be noted that it is not necessary for each of the pipelines 56-59 to operate in only one mode of operation. For example, it is possible for the pipelines 56-59 to operate in both the optimization mode and the jitter mode. As an example, the region 249 could be divided into two portions according to the techniques described herein for the optimization mode. The pipelines 56 and 57 could be responsible for rendering graphical data within one portion of the region 249, and pipelines 58 and 59 could be responsible for rendering within the remaining portion of the region 249. Furthermore, the pipelines 56 and 57 could render jitter enhanced and/or anti-aliased images within their portion of region, and pipelines 58 and 59 could render jitter enhanced and/or anti-aliased images within the remaining portion of region 249. The modes of pipelines 56-59 may be mixed according to other combinations in other embodiments.

Furthermore, it is not necessary for the application 17 to be aware of which mode or combination of modes are being implemented by pipelines 55-59, since the operation of the application 17 is the same regardless of the implemented mode or combination of modes. In other words, the selection of the mode or modes implemented by the pipelines 55-59 can be transparent to the application 17.

It should be noted that there are a variety of methodologies that may be employed to enable the selection of the mode or modes performed by the system 50. In the preferred embodiment, a user is able to provide inputs via input device 115 of client 52 (FIG. 4) indicating which mode or modes the user would like the system 50 to implement. The client 52 is designed to transmit the user's mode input to master pipeline 55 over LAN 62. The slave controller 261 of the master pipeline 55 (FIG. 5) is designed to then provide appropriate input to each slave pipeline 56-59 instructing each slave pipeline 56-59 which mode to implement based on the mode input received from client 52. The slave controller 261 also transmits control information to compositor 76 via connection 331 (FIG. 3)

indicating which mode is being implemented by each pipeline 56-59. The compositor 76 then utilizes this control information to appropriately process the graphical data from frame buffers 76, as further described herein. There are various other methodologies and configurations that may be employed to provide the slave pipelines 56-59 and/or compositor 76 with the necessary mode information for enabling the pipelines 56-59 and compositor 76 to operate as desired. For example, the control information may be included in the data transmitted from the master pipeline 55 to the slave pipelines 56-59 and then from the slave pipelines 56-59 to the compositor 76.

It should be noted that master pipeline 55 has been described herein as only rendering 2D graphical data. However, it is possible for master pipeline 55 to be configured to render other types of data, such as 3D image data, as well. In this regard, the master pipeline 55 may also include an OGL daemon, similar to the OGL daemon 205 within the slave pipelines 56-59. The purpose for having the master pipeline 55 to only execute graphical commands that do not include 3D image data is to reduce the processing burden on the master pipeline 55, since the master pipeline 55 performs various functionality not performed by the slave pipelines 56-59. In this regard, executing graphical commands including only 2D image data is generally less burdensome than executing commands including 3D image data. However, it may be possible and desirable in some implementations to allow the master pipeline 55 to share in the execution of graphical commands that include 3D image data. Furthermore, it may also be possible and desirable in some implementations to allow the slave pipelines 56-69 to share in the execution of graphical commands that do not include 3D image data (*e.g.*, commands that only include 2D graphical data).

In addition, a separate computer system may be used to provide the functionality of controlling the graphics pipelines. For example, FIG. 13 depicts another embodiment of the